

510.84

116r

no. 806

cop. 2

ILLINOIS UNIVERSITY DEPARTMENT
OF COMPUTER SCIENCE

REPORT

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

To renew call Telephone Center, 333-8400

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

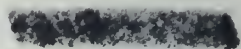
FEB 12 REC'D

FEB 15 2001

UNIVERSITY OF
ILLINOIS LIBRARY
AT URBANA-CHAMPAIGN

DATE RECEIVED

L161—O-1096



510.84
IL6r
cop.2
#806

Report No. UIUCDCS-R-76-806

NSF-OCA-DCR73-07980 A02-000021

ANALYSIS OF COMPUTER ARCHITECTURES
FOR INFORMATION RETRIEVAL

by

Bernard John Hurley

May 1976



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS



Report No. UIUCDCS-R-76-806

ANALYSIS OF COMPUTER ARCHITECTURES
FOR INFORMATION RETRIEVAL*

by

Bernard John Hurley

May 1976

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. US NSF DCR73-07980 A02 and was submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, May 1976.

ACKNOWLEDGMENT

While studying at the University of Illinois, I have made many friends who have encouraged my academic and social growth. I would like to thank my thesis advisor, Dr. Duncan Lawrie, for the criticism and suggestions offered in our conversations spanning the past two years. The advice and support of Dr. David Kuck and Dr. William Stellhorn was greatly appreciated. I would also like to thank the EUREKA people, Mr. Dick Rinewalt, Mr. Keith Morgan, and Mr. Mike Milner, for their aid and friendship.

Many other people who have not been directly involved in my work have been instrumental in my emotional well-being. I would like to thank Mrs. Virdi Devore, Mrs. Ann Ryczer, Mr. Ben Rund, and Mr. Roy Nelson, for the coffee and conversation that kept me working while the rest of the world slept. I am in debt to all who have offered me their friendship during my stay at Illinois.

Finally, I would like to give special thanks to my parents and family. Your support has never been lacking.



Digitized by the Internet Archive
in 2013

<http://archive.org/details/analysisofcomput806hurl>

TABLE OF CONTENTS

	Page
1. INTRODUCTION.....	1
1.1 EUREKA Project.....	1
1.2 Background and Goals.....	2
2. SYSTEM ARCHITECTURE.....	4
2.1 Basic Architecture.....	4
2.2 Secondary Storage Subsystem.....	4
2.3 Processor Subsystem.....	6
2.4 Control Subsystem.....	7
3. ARCHITECTURAL DESIGN VARIABLES.....	9
3.1 Secondary Storage Devices.....	9
3.2 Transfer Bandwidth.....	10
3.3 Processor Bandwidth.....	11
3.4 Main Memory Size.....	11
4. SINGLE USER SIMULATIONS.....	13
4.1 Simulation Specifications.....	13
4.2 Simulation Results.....	13
4.3 Summary of Important Results.....	20
5. MULTI-USER SIMULATIONS.....	23
5.1 Multi-user Goals and Definitions.....	24
5.2 Multi-user Performance Measures.....	26
5.3 Varying the Job Mix.....	29
5.4 Data Base Pre-seeking.....	35
5.5 Main Memory Constraints.....	38
5.6 User Response Times and Search Arrival Rates....	40
6. CONCLUSIONS.....	50
LIST OF REFERENCES.....	52

1. INTRODUCTION

1.1 EUREKA Project

Automated information retrieval is a rapidly developing discipline concerned with the speed and accuracy in which information needs can be fulfilled. At the University of Illinois, Urbana, we are engaged in a wide based program to study some of the problems which are realized in a large, on-line, text retrieval system [1]. The subjects under consideration include hardware, software, and human factors which may influence system performance. Our efforts are centered around developing EUREKA, a mini-computer based retrieval program, into an efficient, easy to use, on-line text retrieval system. Utilizing EUREKA as a research tool, we will be able to study how users adapt to an on-line environment as well as develop search strategies. Experiments conducted with EUREKA will also help to determine the importance of special software features supplied to aid the user in his/her efforts [1,2]. Another area of interest concerns reducing search response time while increasing system throughput. To help attain this goal we are studying data structures, experimenting with different levels of inversion [3] and designing special processors [1,4,5] to upgrade EUREKA'S performance.

ty/argh the
it's not the
spit it
off our
studies are
in
Base on
our studies are.

At present one can find many software packages [6], such as EUREKA, designed to fulfill the function of storing and recalling information. As retrieval systems become more popular and are forced to handle larger data bases it is reasonable to think beyond software systems to a computer designed for this type of processing. This paper is the result of an effort to support and supplement the development of EUREKA by studying and evaluating different machine architectures which are suitable for information retrieval.

1.2 Background and Goals

We can begin our discussion on specialized retrieval computers by defining the structure of the data to be searched. We will confine our discussion to text retrieval using inverted files although the principles are applicable to other types of data. Large data bases require many index terms to be used in search requests as access points into the data. For every index term there will exist a postings list which contains the identifiers of all books, journals, paragraphs, sentences, etc., containing that particular index term. Coordinating these terms using Boolean operators will require comparing the postings lists as we scan for the identifiers that satisfy the search request. The list of identifiers which is the answer to the request will be known as

the resultant list of that search. If there are more than two index terms being coordinated we will have intermediate resultant lists. These are lists which were produced by comparing two postings lists but still need to be matched with others to form a final answer.

A specialized machine architecture for information retrieval must overcome the problems found in conventional computers used for this type of non-numeric processing. The task of comparing postings lists is very time consuming, especially if these lists are long or many such lists need to be matched. To reduce this problem, a hardware unit could be built to perform the merging and Boolean coordination necessary to complete the comparison operation. Designs of such units have already been proposed in [4] and [5]. Along with these faster comparison networks, it may be necessary to increase the speed at which postings can be moved through the system. The reason for this being, the need to keep the comparison hardware supplied with data to process.

In the rest of this paper we will consider architectures which can both move and process posting lists at different rates. Our goal will be to identify important relations between these machines and the components of which they consist. With this knowledge we will be better prepared to propose an architecture specially designed for information retrieval purposes.

2. SYSTEM ARCHITECTURE

2.1 Basic Architecture

A system architecture from which different retrieval machines can be configured [3] is shown in Figure 2.1. A brief explanation of the function and construction of each subsystem is in order.

2.2 Secondary Storage Subsystem

First we can consider the secondary storage subsystem which includes the disk units and controller. The types of disk units available are head-per-track and moving arm. In the former version there exists a read/write head for every physical track on the disk. The latter has its read/write heads mounted on an arm which is placed on the cylinder with the appropriate data. To facilitate fast data movement through the machine, the secondary storage subsystem will be allowed to transfer up to 'M' postings in parallel per hardware cycle. We will define a posting to be one computer word long and the length of a hardware cycle to be the time to read or write one word of data (one posting) from the disk using one read/write head. The disk control unit will consist of three independent channels each of which is capable of

A powerful operation can be performed if we let 'M', the number of postings moved from secondary storage per hardware cycle, equal 'N'. Using the aforementioned ability to read from one channel while writing on another, we can simultaneously read a postings list from secondary storage, process it with another list already in main memory and, if desired, write the resultant list back to disk. The total time for this operation will be slightly more than if both lists originated, and the resultant remained, in main memory.

2.4 Control Subsystem

The control subsystem is the last architectural component to discuss. Here exists a control unit which is responsible for the allocation of system resources as well as supervision of the control algorithm. To solidify the function of each subsystem and demonstrate how they interact we can briefly describe one possible algorithm. The events of this control algorithm are:

- (1): If two postings lists reside in main memory and the merge-coordination network is not busy, they are processed.
- (2): If the merge-coordination network is free and there is only one list in main memory, a postings list is read

from secondary storage and simultaneously processed with the one in memory. The resultant list is kept in main memory.

(3): If the merge-coordination unit is busy the secondary storage subsystem may read a postings list into main memory.

(4): When main memory becomes full:

(a): All the lists in the main memory are processed to form one large intermediate resultant list.

(b): This list is then processed simultaneously with a large list from secondary storage. The resultant list is written to disk as the two original lists are being processed.

The events described are repeated until there remains one final resultant list. At this point the search has been completed.

3. ARCHITECTURAL DESIGN VARIABLES

Using the architecture described in the previous section there are at least four design variables which may be manipulated. A change in any one of these can result in a new machine with its own advantages as well as drawbacks. These variables are:

- (1): Device type
- (2): Transfer Bandwidth
- (3): Processor Bandwidth
- (4): Main Memory Size

We could consider a fifth variable, search control and scheduling, which also will affect system performance. This software related area will not be pursued in this paper but has been studied in [7].

3.1 Secondary Storage Devices

Device type refers to the class of secondary storage unit being utilized. We will consider both moving arm and head-per-track disks. The elimination of all seek time in the latter type would lead us to believe it is a better choice

with respect to system performance. The obvious advantages of the moving arm disk lie in its low cost and proven reliability.

3.2 Transfer Bandwidth

The second design variable, transfer bandwidth, is a measure of the amount of data moved by the secondary storage system per hardware cycle. A system which can transfer sixteen postings in parallel within one hardware cycle is clearly more powerful than one which can transfer less. We would expect the transfer bandwidth to influence two major areas. The first area concerns the secondary storage's ability to provide an adequate flow of postings to the main memory for processing. The second area is the speed at which intermediate resultant lists, that will not fit in main memory, can be returned to disk. Increasing the transfer bandwidth allows data to be moved quickly through the system, thus upgrading performance. To obtain this large bandwidth we must modify conventional secondary storage units to transfer in parallel. The trade-off we experience balances the improvement in search times acquired while using a large transfer bandwidth against the difficulty and cost of modifying the secondary storage subsystem. We must also consider the reliability of the modified hardware units.

3.3 Processor Bandwidth

Processor bandwidth measures the number of postings which can be input or output from the processor sub-system in one hardware cycle. The effect of implementing a large processor bandwidth will be to shorten the time it takes to form resultant lists. The processor bandwidth should be balanced with the transfer bandwidth to allow efficient movement of data through the entire machine. An architecture with a processor subsystem that is more sophisticated than necessary will impose stringent timing requirements and higher costs that could have been avoided.

3.4 Main Memory Size

The last design variable is main memory. This will be an important factor because it influences the number of times an intermediate resultant list must be temporarily placed back in secondary storage. If a particular architecture utilizes slow I/O devices this memory clearing operation can become a time consuming operation.

When designing a specialized architecture for information retrieval the goal would be to balance the subsystems for maximum performance and efficiency. For a given processor bandwidth, the variables which influence secondary storage,

disk type and transfer bandwidth, should be regulated to keep the processor supplied with useful data. The main memory size must be large enough not to degrade the machine by forcing the secondary storage subsystem to wait for space to store its data for the processor. A small main memory will also have the detrimental effect of increasing the number of times intermediate resultant lists must be cleared to disk. The question of subsystem balance with respect to the design variables is one that must be answered if efficient machines are to be built.

4. SINGLE USER SIMULATIONS

4.1 Simulation Specifications

A simulation program was employed to provide performance figures for different values of the design variables listed in Chapter (2). With this data we will be better able to understand the relationships between these variables and their influence on an architecture as a whole. The search we simulated consisted of seventy index terms related by the Boolean 'OR' operator. These terms, as well as the lengths of their postings lists, were taken from the MEDLARS Retrieval System [8] and assigned addresses on the secondary storage units of our simulator. The seventy term 'OR' was used as our benchmark because it is typical of the more time consuming operations found in a system such as MEDLARS. The search simulation was started and allowed to run to completion using the control algorithm described in Chapter (2).

4.2 Simulation Results

Initially there were eight architectures studied in three main memory sizes. Table 4.1 describes the architectures which, for convenience, have been labeled (a) to (h). Another

Architecture Name	Processor Bandwidth	Data Base Storage		Temporary Storage	
		Disk Type	Transfer Bandwidth	Disk Type	Transfer Bandwidth
a	L	HT	L	HT	L
b	L	HT	S	HT	S
c	S	HT	L	HT	L
d	S	HT	S	HT	S
e	L	MA	L	HT	L
f	L	MA	S	HT	S
g	S	MA	L	HT	L
h	S	MA	S	HT	S
i	L	MA	S	HT	L

Key:

- L = Large Bandwidth
(16 postings/hardware cycle)
- S = Small Bandwidth
(1 posting/hardware cycle)
- HT = Head-per-track Disk
- MA = Moving Arm Disk

Table 4.1. Architectures

system, (i), was added later and will be discussed separately. The results obtained from the simulation of these architectures can be found in Figure 4.1. The following three rules can be used as a guide for interpreting the graphs formed for each memory size in Figure 4.1.

- (1): We will define the front plane of each graph to include Architectures (a), (b), (c) and (d). These are all machines which use head-per-track disks as secondary storage units. The back plane, Architectures (e), (f), (g) and (h), utilize disks which are moving arm.
- (2): The central axis of each graph contains Architectures (a), (e), (d) and (h) all of which represent machines that have equal processor and transfer bandwidths. Architectures (a) and (e) have bandwidths of sixteen postings per hardware cycle while Architectures (d) and (h) have a processor and transfer bandwidth of only one posting per cycle.
- (3): Points off the central axis are machines in which the processor and transfer bandwidths differ. Architectures to the right of the axis use a transfer bandwidth of one and a processor bandwidth of sixteen postings per hardware cycle. Architectures to the left of the central axis have these values for transfer and processor bandwidths switched.

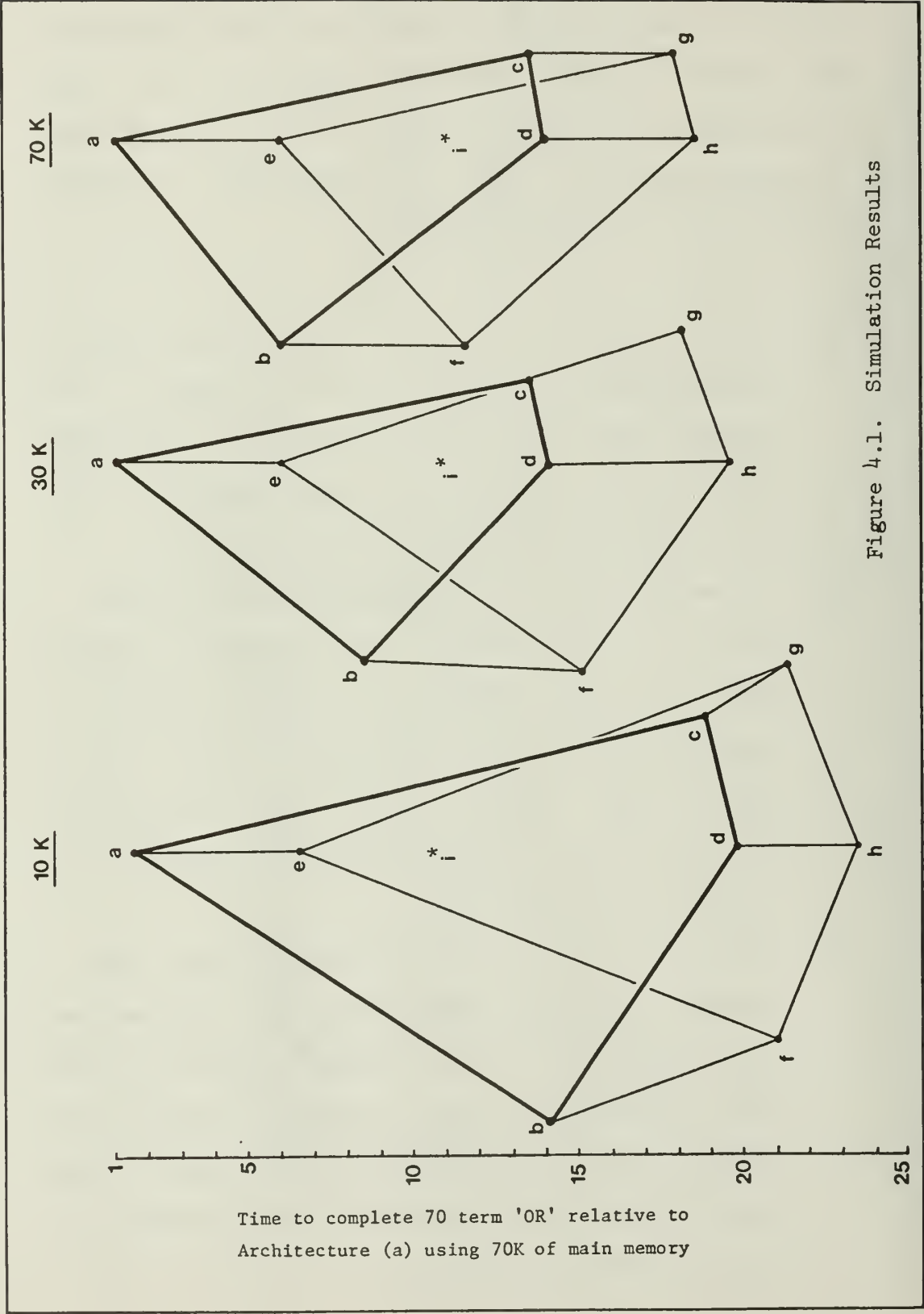


Figure 4.1. Simulation Results

The vertical distance in Figure 4.1 is a measure of the time required by each architecture to complete the seventy term 'OR' relative to the fastest architecture, (a), using seventy kilowords of main memory. Movement that follows a solid line between any two points represents two distinct architectures that change by only one design variable. Therefore, the vertical component of the distance between points is used as a ranking for judging the effects of design variables on an architecture. The only exception to this rule is for the main memory size variable. To study its influence on a machine we must compare across the three graphs.

When studying Figure 4.1 we can identify relations between the eight architectures. For every memory size, all other factors being held constant, head-per-track disks perform better than moving-arm disks as secondary storage units. By eliminating seek time when using the former disk type we have allowed postings to reach main memory with less delay.

For the purpose of discussing processor and transfer bandwidths, let us assume we begin with a machine for which both these variables are small, one posting per hardware cycle. If we enlarge the transfer bandwidth only, the effective decrease in time to complete the search is small. This can be observed in the transitions from Architectures (a)

to (c) and (h) to (g). Enlarging only the processor bandwidth, as in Architectures (b) and (f), results in a machine that displays better performance but is sensitive to changes in the main memory size. Increasing the bandwidth for both design variables simultaneously yields the best search times and is unaffected by fluctuations in the size of main memory. Architectures (a) and (e) have the large processor and transfer bandwidths of sixteen postings per hardware cycle.

Using our newly obtained knowledge on the relations between design variables, we can postulate a new architecture to be labeled (i). The description of this machine, found in Table 4.1, includes the use of a large processor bandwidth. By observing the degradation of Architectures (b) and (f) as the main memory size is reduced, we realize that a machine with a small transfer bandwidth suffers as it is forced to clear more intermediate resultant lists to secondary storage. This effect is primarily due to the amount of time and resources needed to process these large lists. To remedy this predicament, Architecture (i) uses a large transfer bandwidth for the channels and disks employed in clearing any resultant lists from main memory. The data base storage and its associated channel retain a small bandwidth.

Architecture (i) becomes very attractive for three reasons, all which are related to present costs and technology. First, for large data bases it is economically unfeasible to use head-per-track disks as secondary storage units. Employing moving arm disks modified to transfer sixteen postings per hardware cycle may also result in high modification costs and uncertain reliability. Architecture (i) uses conventional moving arm disk units to hold the entire data base. This particular disk has in the past proven its reliability and can be purchased at a relatively low price. Secondary storage units with a large transfer bandwidth will be used by Architecture (i) to load and store intermediate resultant lists from main memory. These units, possibly head-per-track disks or shift register memory, would be few in number and not a large burden to system cost or reliability. Secondly, Architecture (i), marked by the '*' in Figure 4.1, is not prone to degrade as memory size is decreased. In fact, a small peculiarity in the control algorithm causes a very slight drop in performance as larger main memory sizes are used. This effect is unimportant because it still leaves us the attractive option of running efficiently with little main memory. Finally, by once again referring to Figure 4.1, we find Architecture (i) approximately half-way between the best and worst machines. That is, the time it takes to complete a search using

Architecture (1) is a little more than twice the time of the best machine and half the time of the worst. This could most certainly be considered reasonable performance for a machine in which the secondary storage system is constructed almost entirely out of conventional devices.

4.3 Summary of Important Results

We can now restate the important relations found in the simulation results.

(1): Head-per-track disks always outperformed moving arm disks as secondary storage units. This is attributed to the elimination of seek time. The effect of switching to the faster disk type on otherwise similar systems is relatively profitable with respect to search completion times.

(2): Listed below in decreasing order of importance are the five combinations of the processor and transfer bandwidth variables.

(A): Enlarging both the processor and transfer bandwidths to sixteen postings per hardware cycle allowed data to flow freely through the machine. As a result we found architectures with these

parameters display excellent search times.

(B): Architectures that used a large bandwidth for the processor and the secondary storage devices involved in handling intermediate resultant lists displayed the next best level of performance. In this system the data base remained on small transfer bandwidth devices. This machine, with its ability to efficiently handle loading and storing intermediate resultant lists, yielded search times approximately midway between the best and worst architectures.

(C): Increasing only the processor bandwidth displayed the next best performance by allowing postings to be quickly moved through the processor subsystem. Unfortunately, this improvement quickly degraded as main memory sizes were reduced. The reason for this will be explained in (3).

(D): Increases in the transfer bandwidth alone proved to be uninteresting. Postings could not be processed in the main memory fast enough to allow any effective decrease in search times.

(E): The combination of a small processor and transfer bandwidth for any given disk type always resulted in the worst performance. This machine is a close approximation to a 'conventional computer' when

used with moving arm disks.

- (3): Varying the main memory size proved to be an interesting experiment. A close relation between this design variable and the size of the transfer bandwidth was uncovered. If large searches (many postings) are conducted in small main memory sizes we found need to clear the memory by writing intermediate resultant lists to secondary storage. This operation, if done with a small transfer bandwidth, is very time consuming and has significant effects on search times. This explains why architectures with large processor and small transfer bandwidths degrade as they are run in decreasingly small memory sizes.

5. MULTI-USER SIMULATIONS

It is now important to examine the performance of our specialized information retrieval computer in a steady state, multi-user environment. With the data from this study we can determine how our system will respond under the various working conditions found in both commercially available and experimental retrieval systems. Due to the large number of machine architectures proposed in the last chapter, along with the high cost of simulation, we have limited our multi-user studies to Architecture (i). The choice of Architecture (i) was based on its high potential for good performance while using only commercially available devices in the secondary storage subsystem. This particular machine configuration can bring together three very important machine design objectives. That is, we can obtain good performance while constructing a machine that is both economically and technologically feasible. The other architectures developed up to this point are not being dismissed as uninteresting and could be studied in more detail at a later time. In fact, the exact same study could be conducted with any other architecture discussed in the previous chapters.

5.1 Multi-user Goals and Definitions

To allow Architecture (i) the option of running many users in parallel, we introduced a variable number of servers into the system. Each server will be assigned a user's search request to process along with the other servers. The server will not leave that particular request until the search is finished. The time which will elapse between the server's receiving a search request and its completion will be known as the service time or execution time. This includes the idle time a particular server encounters when other servers have control of the system resources. The time a search request must remain waiting for a server to become available will be the queue wait time. Finally, user response time is the sum of service and queue time. This corresponds to the search request response time seen by the users. The following simple formula shows the relationship between all the times we have defined.

$$\begin{array}{lll}
 \text{User Response Time} & = & \text{Queue Wait Time} + \text{Service Time} \\
 \text{(response time seen} & & \text{(time a search} & & \text{(idle time +} \\
 \text{by the user)} & & \text{waits for a} & & \text{processing} \\
 & & \text{server)} & & \text{time)}
 \end{array}$$

Any other new descriptive terms introduced in this chapter will be used, unless otherwise defined, with the standard

queueing theory interpretation.

The object of the multi-user study is to measure performance under different but realistic restraints. More specifically, we ran simulations to determine the effects on performance created by:

- (1): varying the job mix to force processing under heavier work loads.
- (2): increasing the number of disks on the data base channel to speed search response time through pre-seeking.
- (3): decreasing the main memory size to substantiate our claim that Architecture (i) can run well with this restraint.

Unless otherwise stated, all simulations were run using Architecture (i) with five secondary storage devices on the data base channel and main memory partitions of seventy kilo-words per user. The sensitivity of the simulation results to the size of these partitions will be studied in Section 5.5. Also, we will work under the assumption that there is an infinite queue of users waiting to be serviced by the system.

5.2 Multi-user Performance Measures

The results for all multi-user simulations will be plotted as in Figure 5.2.1. On the 'X' axis the number of servers allowed to run in parallel, for a particular job mix using Architecture (i), has been varied from one to eight. For each point on the 'X' axis and an associated job mix, there are two measures of system performance plotted in units of time relative to the 'Y' axis. The result is two performance curves for each job mix which vary in time as the number of parallel servers is changed. We can now further explain these curves.

The top curve is a measure of the mean service response time experienced by an average user. That is, the average time to finish execution of a search excluding any time waiting in queues for a server to become available. Remember, by not including the queue wait time we are only observing execution response which is not the same as the user response time. Mathematically, the mean service response time can be represented as the sum of 'N' users service times divided by 'N'.

$$\text{Mean Service Response Time} = \frac{\sum_{i=1}^N \text{Service Time}(i)}{N}$$

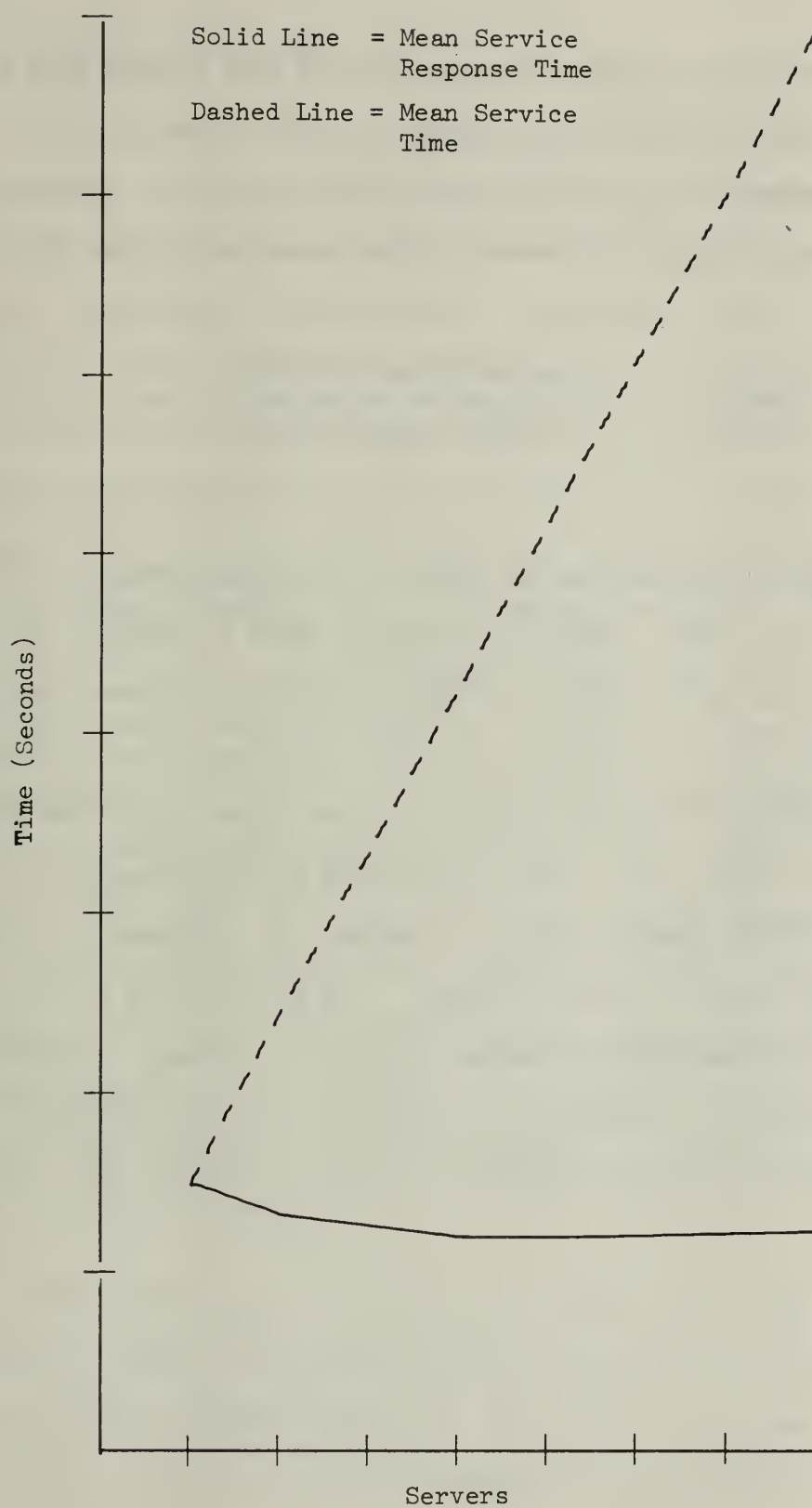


Figure 5.2.1. Performance Curves

The second curve is a measure of the mean service time in terms of the average time to serve a search request. This can be expressed as a particular finite run time of the retrieval machine divided by the number of jobs processed in that time.

$$\begin{array}{rcl} \text{Mean} & & \text{Finite System Processing Time} \\ \text{Service} & = & \text{-----} \\ \text{Time} & & \text{No. of Jobs Processed} \end{array}$$

It is important not to confuse the two performance curves which are both plotted in terms of units of time per user. Suppose we had a system with three servers and three search requests which finished in three, six and nine seconds respectively. The mean service response time would be six seconds while the mean service time is three seconds. The mean service response time is a measure of response to the average user's search request. The mean service time is a measure of how fast the system is working. It may be easier to think of our definition of mean service time as the inverse of the more standard definition of 'jobs processed per unit time'.

5.3 Varying the Job Mix

In realistic retrieval installations we may find many different types and sizes of searches. As the search queries become larger they demand more of the retrieval computers total processing resources and consequently have larger response times. For this reason, we decided it was necessary to simulate different job mixes and determine what effect the larger searches would have on the individual user and the system as a whole. The first job mix we studied uses information gathered in [9] on the MED-LINE retrieval system. Using this data we chose the number of index terms in a search to be uniformly distributed with an average of four terms. The length of a postings list was exponentially distributed with an average of six hundred postings to an index term. Simulations were then run for three more job mixes using averages of eight, sixteen and thirty-two index terms per search. The larger number of search terms found in the latter simulations can pose a realistic study if one allows for the possibility of an automatic thesarus to aid the user in selecting these terms.

When analyzing the results in Figure 5.3.1 we find definite trends in the data. First, let us examine the mean service response time curves for the four job mixes. As expected, the mean service response time increases as more

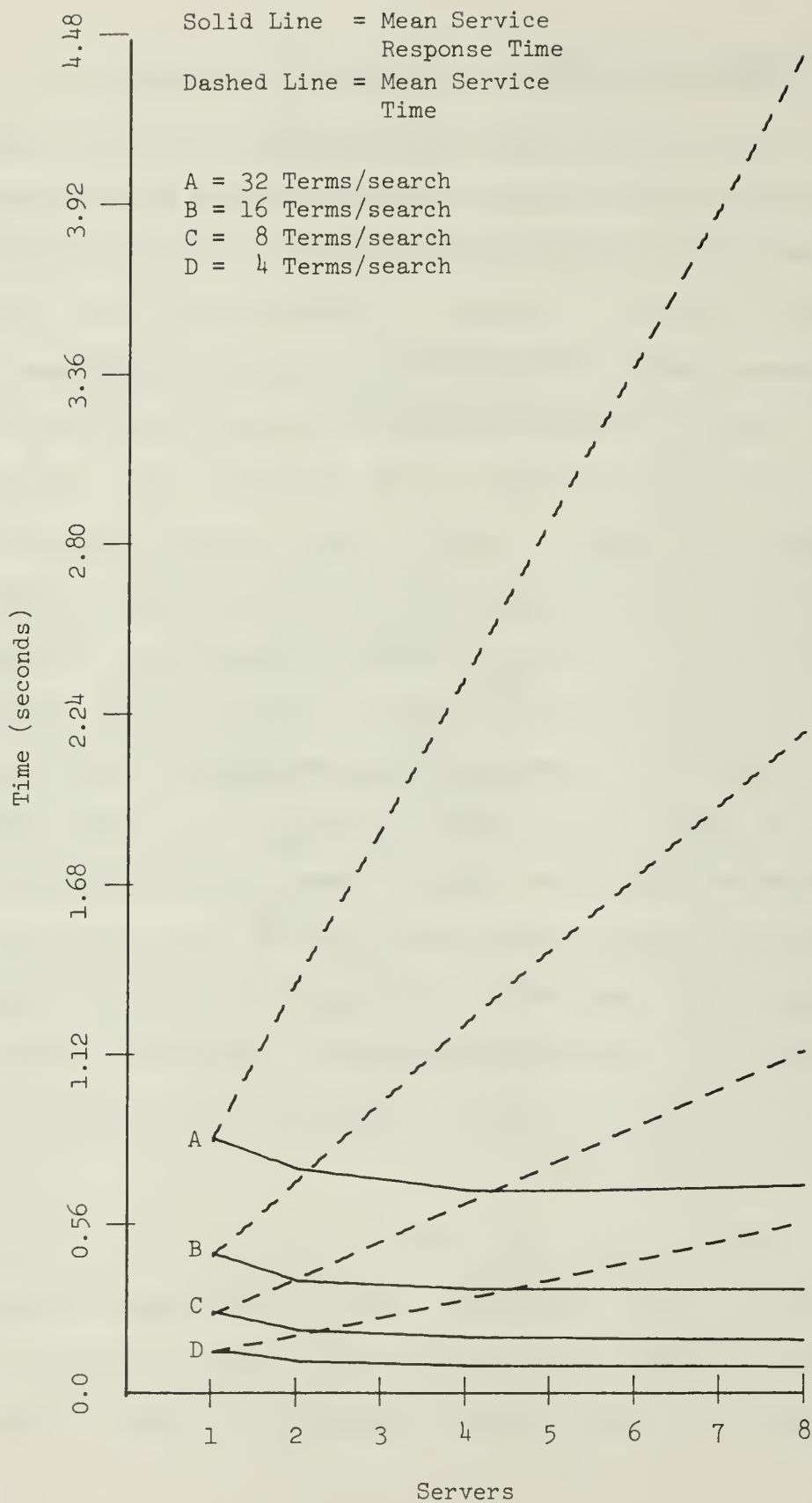


Figure 5.3.1. Job MIX Performance

servers are allowed to run in parallel on the system. This is due to the competition among jobs for the systems resources. More interesting is the increase in the slope of the mean service response time curve as the average number of index terms per search grows. When studying the eight server case from Figure 5.3.1, we will observe the increase in time from four to thirty-two terms per search is a factor of eight; the same as the increase in terms. As the number of servers is decreased, we find the mean service response time improving and becoming less related to the change in the number of terms. Once again, this can be explained by the competition between jobs for resources. An example may help clarify this point. In the steady state, a system with eight servers and an average of four search terms per request must retrieve and merge approximately thirty-two postings lists for these jobs. A job mix with an average of thirty-two terms would process two hundred fifty-six lists. This increase in the number of postings lists becomes important because it will determine two important factors. First, the number of times the disk arm must stop during one scan of the data is dependent on the number of lists to be retrieved. As we increase the number of lists to read, the time one search waits to have its lists served will also grow. Second, as more lists arrive in main memory, the queue for the special processor will become longer and also cause an increase in service time. Allowing more

servers or larger searches in the system will have a very degrading effect on the small searches. The execution time of these searches will be grossly inflated as a result of the time they were waiting for disk or processor service. This in turn will raise the mean service response time considerably. The rise in the slope of the mean service response time curve can then be attributed to the degradation of disk and processor service time experienced by smaller searches as the average number of search terms or number of servers is increased. It is important to note that even in the worst case, increasing the average number of terms per search while using eight servers, we still only add to the mean service response time by a factor approximately equal to the increase in terms. Before making any more performance judgements on the mean service response time, let us first examine the highly related mean service time.

Since mean service time is also measured in terms of 'time per search', we find that the curve initially falls as servers are added. This can be easily explained by the increase in system utilization which occurs when one allows more than a single server to process searches. To understand fully the change in mean service time at any point on the curve, it is often helpful to compare the specific point to the single user case. By doing this, we can observe if the

average time per search has fallen due to sharing of resources or has been increased because the sharing has become a degrading competition. A quick example will show that for all job mixes, the average time per search is better for systems with two servers than for a single server. This means that the same work load can be processed in less time by two servers because they share the special processor and secondary subsystem to upgrade their respected utilization factors. When comparing mean service time across the various job mixes for systems with one to four servers, we find a decrease in mean service time. This trend is more dominant for large job mixes because they have more work to share between the system resources. As we increase the number of servers past four, we find small job mixes show little improvement in mean service time while large job mixes are actually doing less work per unit time. Why do we find the mean service time degrading? Once again, it is the small searches waiting long periods of time to have their postings lists read and processed. These searches 'clog' the system by remaining active while they are essentially doing nothing.

To gain a complete understanding of the data represented in Figure 5.3.1, for each job mix, we must look at the mean service response time and mean service time curves together. It would be desirable to operate our retrieval computer using

the number of servers which optimized mean service time for a particular job mix. We may run at this point only if the associated mean service response time is also acceptable. For example, using an average of thirty-two terms per search, we see from Figure 5.3.1 that mean service time is best with four servers. The mean service response time at this point is an acceptable 0.663 seconds. We are saying then, for this job mix with a minimum mean service time, the response time for a search request, ignoring any time waiting for a server to become free, is within reason. At this point, we are not yet well enough equipped to answer the more interesting question of 'What will the response time be if we do include time spent waiting for a server to become free?'. This will be studied in Section 5.6. What we can summarize from our above discussion is this:

- (1): As expected, the mean service response time increases as we add servers to help process a given job mix. This is due to the competition between searches for the system resources. The slope of the curve can be thought of as a measure of the amount of competition or interference between the searches.
- (2): As we add to the average number of terms in a search, we find the mean service response time curve shifts up and also becomes steeper. This is again caused by the

added competition for system resources. It is important to note that in the worst case, with eight servers, the increase in time is only proportional to the number of terms added to the average search.

- (3): Mean service time for any job mix decreases as we begin to add servers to our system. This is due to a constructive sharing of resources which raises their utilization. At some point, the sharing of resources becomes competition and the improvement decreases.
- (4): For any job mix, we can find the number of servers which will optimize mean service time. For this point we can also find the related mean service response time. In a later section we will consider the trade-offs which may arise between response time seen by the user and the mean service time.

5.4 Data Base Pre-seeking

A standard means of increasing the data flow between the secondary storage devices and main memory is to allow pre-seeking among the data base disks. This method can be advantageous if there are many postings lists to read and they are evenly distributed across the disks. Simulations using Architecture (i) and a job mix with an average of thirty-two

index terms per search were run to determine the effect of pre-seeking on our specialized retrieval computer. In Figure 5.4.1 we have plotted the data mean service response times and mean service time curves for systems using one, five and ten disks on the data base channel. The difference between one and ten pre-seeking disks, for both performance curves, is not quite a factor of two. Between five and ten disks there is only a very small speed up. From this data we can conclude that adding more than five disks on a channel will not improve performance greatly. The reason for this can be traced to the saturation of the channel. That is, the channel cannot transmit data as fast as the disks are finding these posting lists.

Apart from the improvement in the performance curves, we can make a second important observation from Figure 5.4.1. The fact that ten disks can be supported by one data channel without performance degradation is indeed encouraging. This means we can minimize the number of relatively expensive data channels needed for the data base. Of course, if more channels were to be added we would again increase the amount of data that could be transferred up to a limit set by the speed of our main memory.

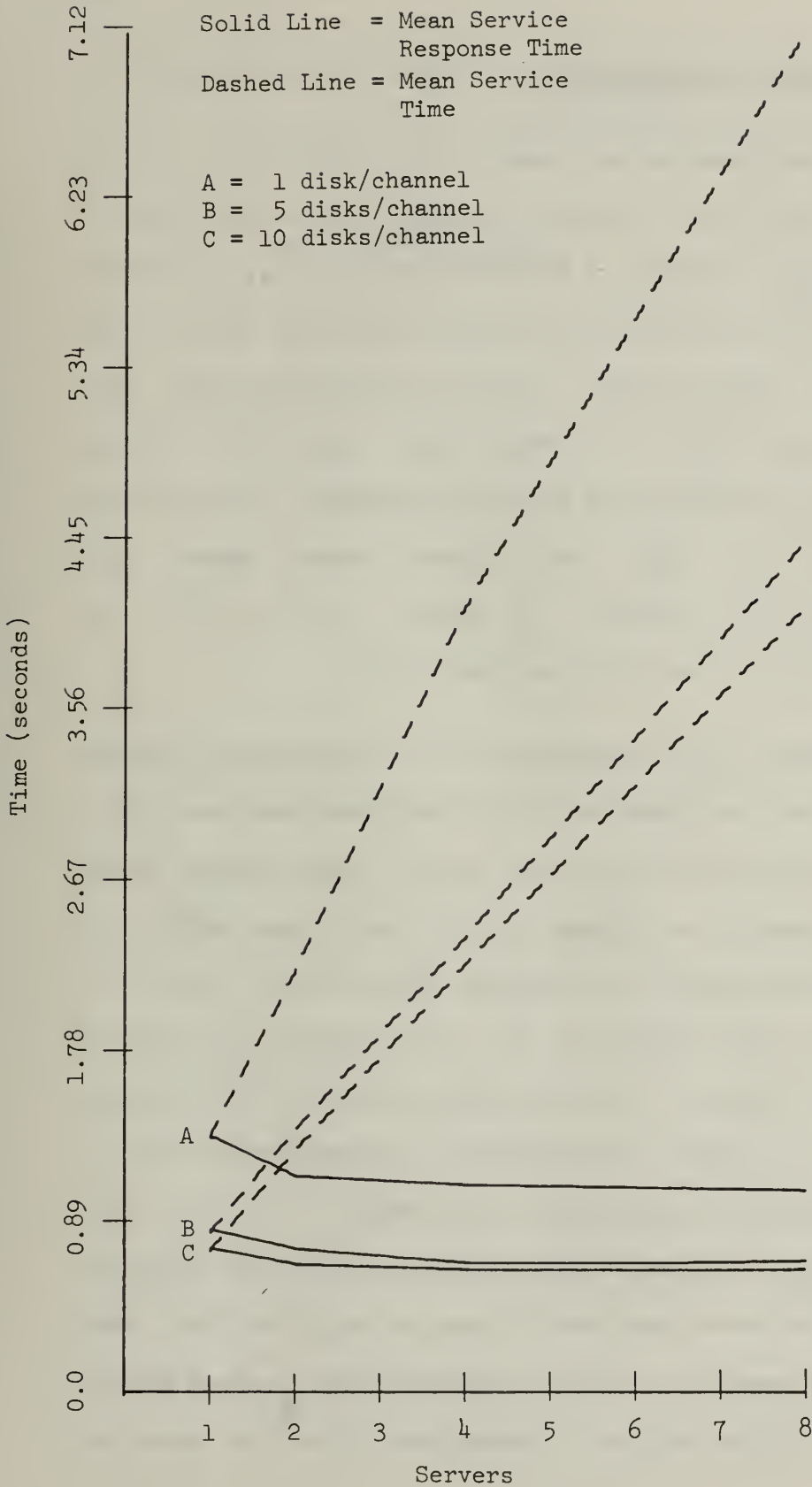


Figure 5.4.1. DISK Performance

5.5 Main Memory Constraints

When implementing multi-users with Architecture (i), we decided to give each server a main memory partition to use as buffer space. The size of the partition, seventy kilo-words, represents much more main memory than was needed by any server to process a search request. Choosing partitions this large insured that the main memory size would not influence performance measures being studied in relation to other design variables. In fact, our largest average search, with thirty-two index terms and six hundred postings per term, would only need twenty kilo-word partitions.

In Figure 5.5.1 we have plotted the mean service response and mean service times for simulations using searches with an average of thirty-two terms and varying main memory sizes. Let us assume, as stated before, that we need only twenty kilo-word partitions for an average search of this size. We then see that decreasing the main memory size to one kilo-word, a factor of twenty, has very little effect on our performance curves. The reason for this impressive result is the specialized storage hardware Architecture (i) uses to hold intermediate resultant lists. The large transfer bandwidth and head-per-track secondary storage devices allow for fast, efficient movement of data between the main memory and the secondary storage subsystem. Consequently, we can dump and

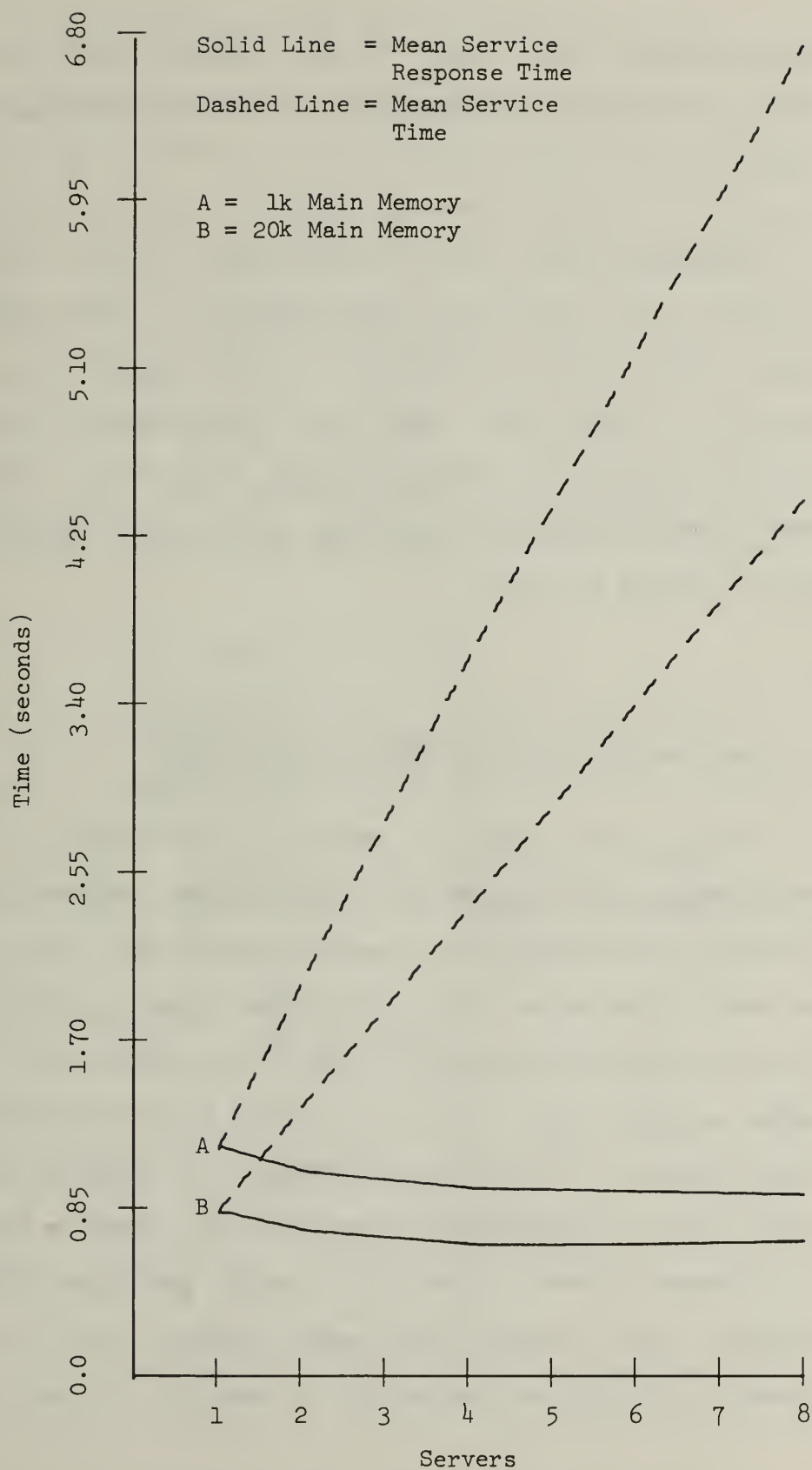


Figure 5.5.1. Main Memory Performance

restore posting lists from the main memory with little penalty. The design of Architecture (i) is fully explained in Chapter (4).

These simulations which vary main memory size verify our prediction that Architecture (i) can perform well using little memory. This result is important not only because it is economical to use less memory but it encourages us to add servers to increase performance through multi-users. Small memory partitions can allow for more searches to be active in the main memory available.

5.6 User Response Times and Search Arrival Rates

So far we have studied two measures of performance, the mean service and mean service response times. Another very important consideration is the search response time seen by the users of the system. The user response time, a sum of the queue wait and search service times, is sensitive to the search arrival rate, the speed at which users are entering search requests. If jobs (search requests) are arriving very fast, they will experience longer queue wait times in their bid to secure a server. The final result is longer user response times. Because the search arrival rate is an important variable, performance will be measured by plotting

the fastest arrival rate which will yield a specific average user response time for a given number of servers. Unless otherwise stated, we will use the term 'user response time' to denote the average of this time over all users. To determine this average arrival rate, we will employ a queueing theory model [10] which assumes poisson arrival and service times, a variable number of servers and an infinite queue to hold users waiting to be served. Figure 5.6.1 is a visual description of this model. The formulas which are used to generate the user response times from the mean service times are as follows:

(1): λ = The search arrival rate

(2): L = Expected number of searches in the system
(waiting and service)

(3): W = Expected user response time

(4): $\mu(k)$ = The mean service time using 'k' servers

(5): $P(n)$ = The probability 'n' searches are being served

$$(6): P(0) = 1 + \sum_{n=1}^N \frac{\lambda^n}{n! \mu(k)}$$

$$(7): P(n) = \frac{\lambda^n}{n! \mu(k)} P(0)$$

$$(8): L = W \lambda$$

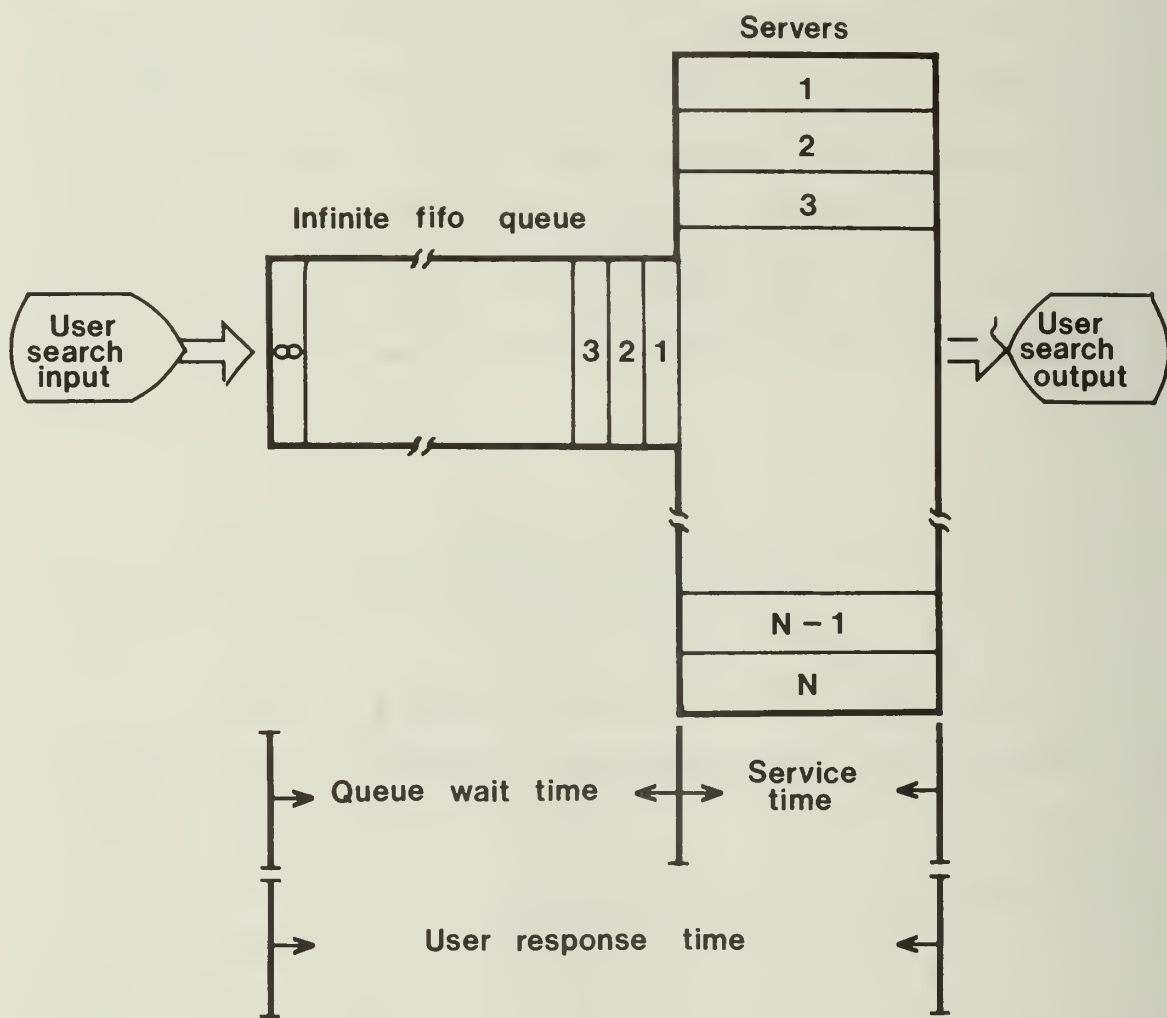


Figure 5.6.1. System Model

In Figure 5.6.2 we have plotted the arrival rate which will yield a two second user response time for the job mixes studied in Section 5.3. Two important facts can be learned from this figure. First, for a particular job mix and a given number of servers, we can observe how fast jobs may arrive while still guaranteeing two second user response time. Second, for each job mix, a specific number of servers can be selected to optimize both cost and performance for this particular architecture. Let us now look at each job mix.

(1): Searches with an average of four index terms.

As servers are added we find a large increase in the arrival rate which yields a two second user response time. This result is expected because it takes many small jobs to fully utilize the system's resources. Because the arrival rate does not peak, begin to drop after the initial increase, we cannot find the maximum rate this system can process. The best number of servers to use would be four. Adding more servers does not increase the potential arrival rate significantly but will increase the main memory needed. Remember, each extra server needs his own main memory partition.

(2): Searches with an average of eight index terms.

Again we find from Figure 5.6.2, that multiple servers increases the search arrival rate for a two second user

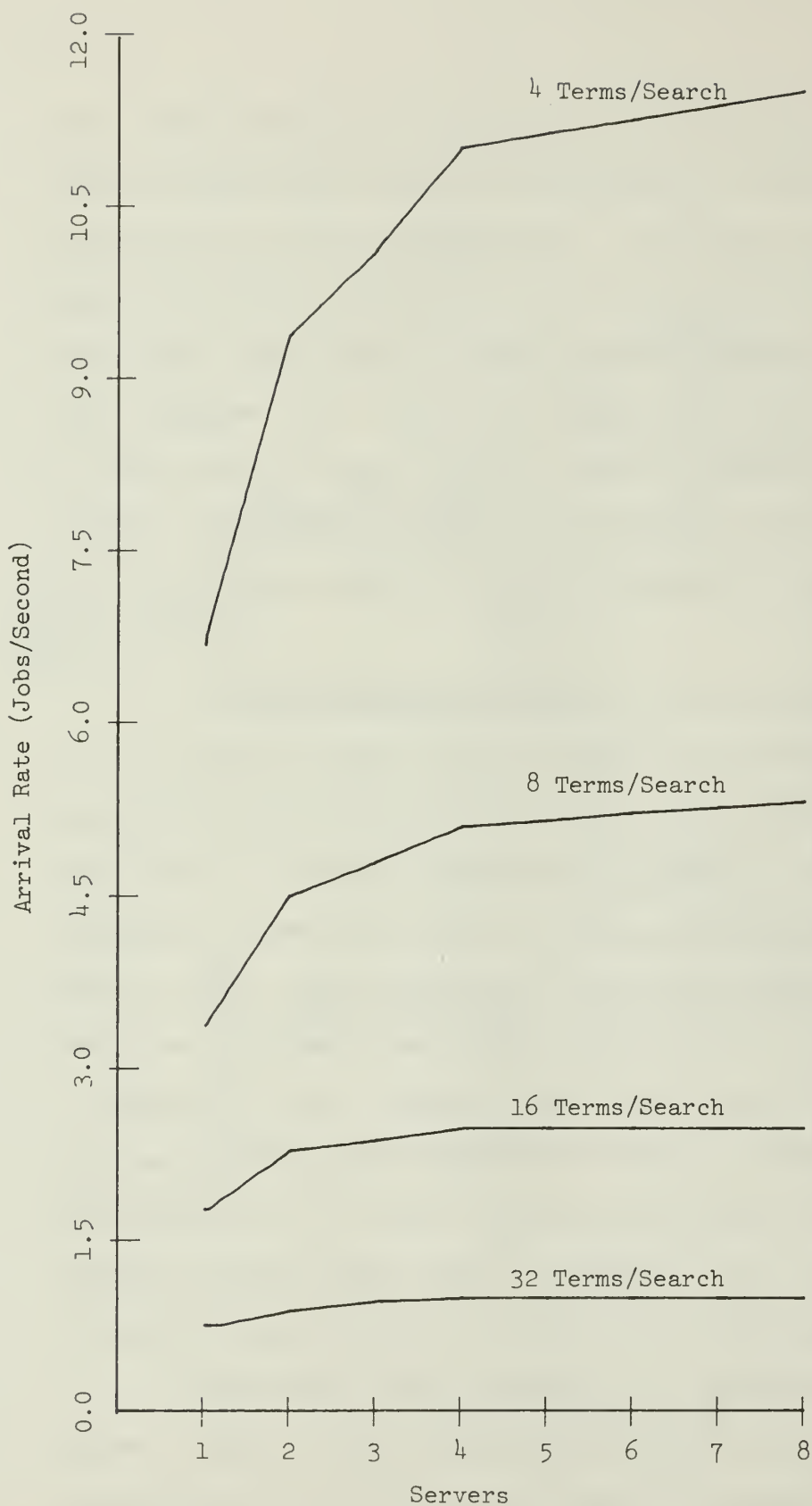


Figure 5.6.2. Arrival Rates for Two Second User Response Time

response time. The change is not as dramatic as above because the searches are larger and require more processing. The arrival rate does not peak and the optimum number of servers is again four.

(3): Searches with an average of sixteen index terms.

Here we find the searches are large enough to hold down much of the improvement gained through multi-servers. The curve which corresponds to this job mix on Figure 5.6.2 does peak at four servers. This means if five or more servers are added, the performance will degrade. The maximum arrival rate that can be processed is then 2.478 jobs per second (four servers) and the optimum number of servers is two.

(4): Searches with an average of thirty-two index terms.

In this case the searches are so large that adding servers is of little help. The search arrival curve peaks at four servers, 0.997 jobs per second, and the optimum number of servers is one, possibly two.

From the above discussion we learned the optimal number of servers and the corresponding search arrival rate for each job mix and a two second user response time. What would happen to performance if we restricted user response to one second? Figure 5.6.3 shows the fastest arrival rates that can be absorbed and still give user response times of one second.

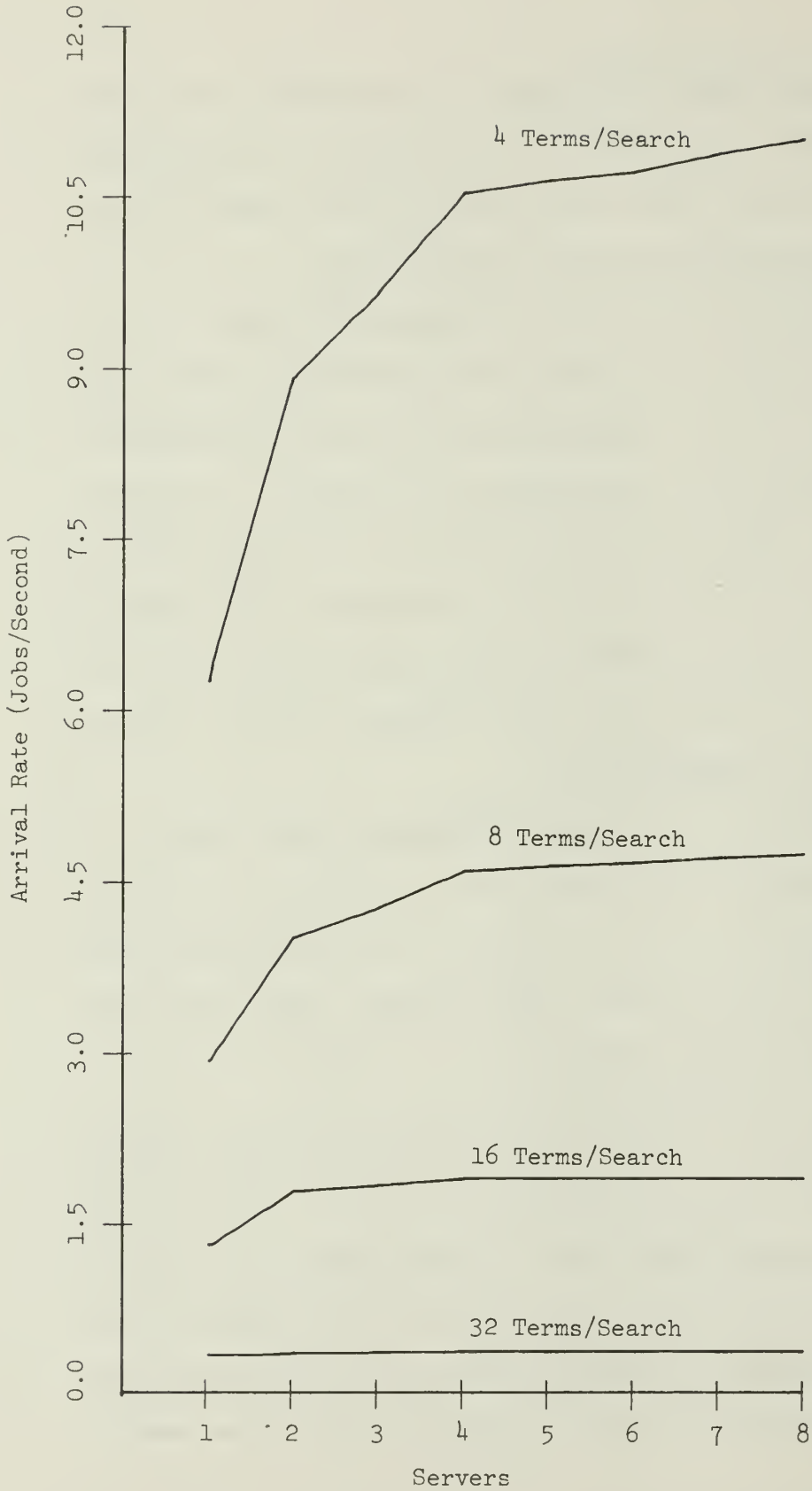


Figure 5.6.3. Arrival Rates for One Second User Response Time

At this point it is not productive to give an in-depth account for each job mix because of the similarity to the curves described in Figure 5.6.2. It is important to note that the decrease in the projected user response time had a very small effect on the search arrival rate that could be supported. This is encouraging because it demonstrates that acceptable user response times are not extremely sensitive to the search arrival rate.

The last study in this section will concern search arrival rates for two second user response, searches averaging thirty-two index terms and special systems. The special systems will include the studies performed in Sections 5.4 and 5.5. Specifically, they are systems with small main memories, one kilo-word, and a varied number of data base disks, one, five and ten.

Figure 5.6.4 shows the search arrival rates for our first special system. Curves are plotted for Architecture (i) using both one and seventy kilo-words of main memory. Since the searches we are using here, thirty-two term averages, need only twenty kilo-words on the average, we find that decreasing the memory by a factor of twenty only decreases the potential arrival rate by about a factor of two. The optimal number of servers to use for a one kilo-word memory is two. After this there is no significant increase in the arrival rate.

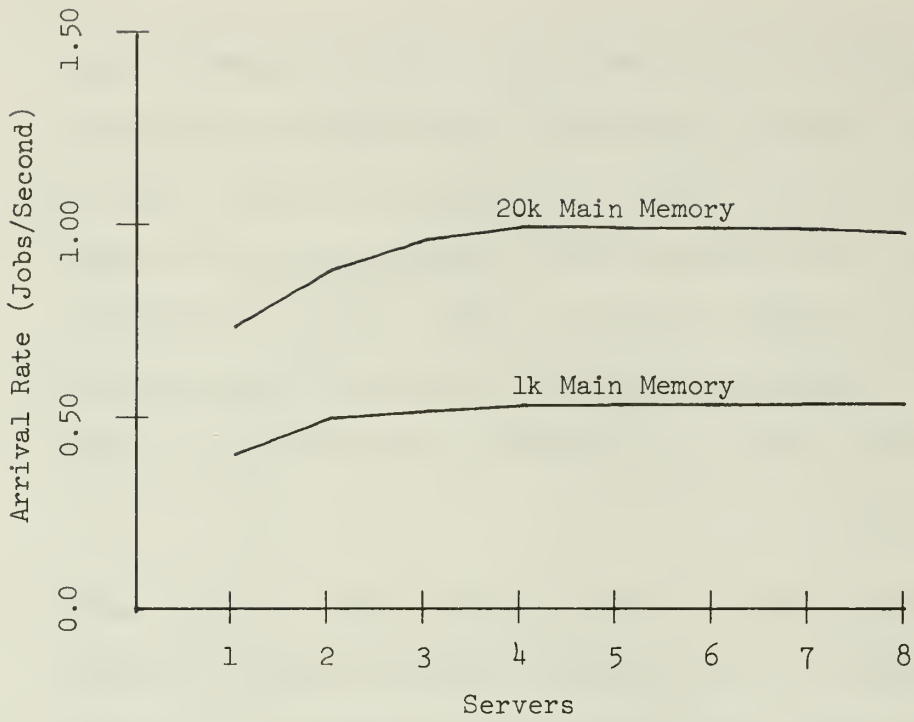


Figure 5.6.4. Arrival Rates for Varied Main Memory Size

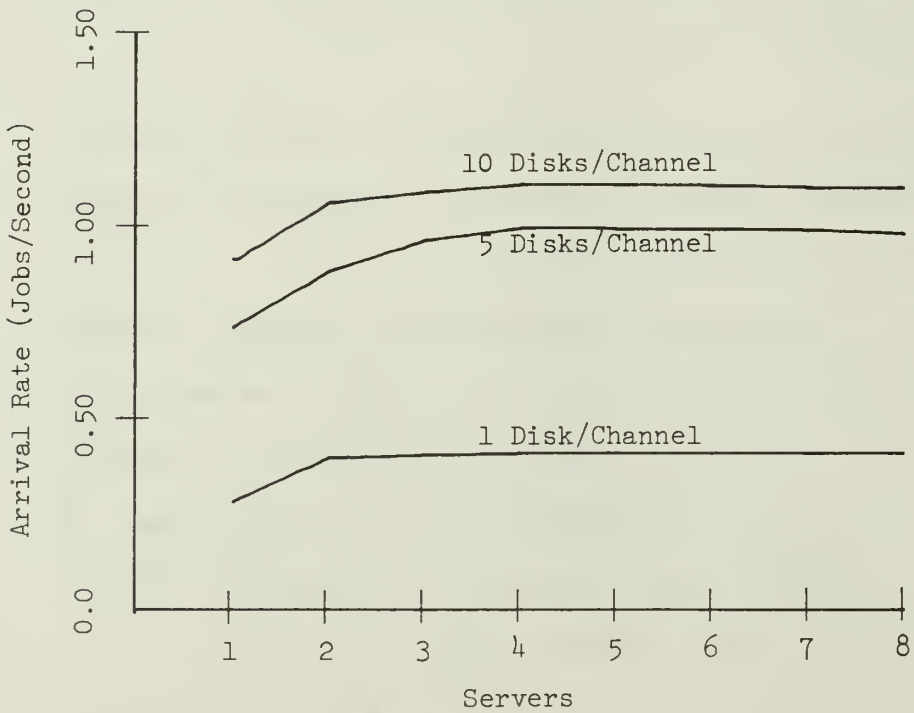


Figure 5.6.5. Arrival Rates for Varied Number of Disks

The second set of special systems has their search arrival rates plotted in Figure 5.6.5. The three curves show the performance for Architecture (1) with one, five and ten data base disks. As in Section 5.4, we find the difference between one and either five or ten data base disks is approximately a factor of two. The increase in performance is attributed to pre-seeking. The details for this argument can be found in Section 5.4. The curves for both one and ten data base disks show that an optimal number of users is clearly two. The search arrival rate data for a system with five data base disks seems to find an optimal cost, performance trade-off with three users.

The search arrival rates plotted in this section are important because they give a measure of performance and help to find the optimal number of servers for a given architecture, user response time and job mix. The study we conducted used Architecture (1) as a basic machine configuration but any other architecture could be analyzed in the same manner. The most important observation in this whole section is that Architecture (1), or any variation of this architecture, performed extremely well by conventional standards.

6. CONCLUSIONS

As the demand for information retrieval services increases so will the need for a specialized machine to perform this function. The basic architecture for this machine should include a processor designed to execute the repetitive task of matching document identifiers contained in postings lists. Also, a memory heirarchy which permits a free flow of data to and from this processor must be included. With this end in mind, a basic architecture was proposed in Chapter (2). From this structure we defined design variables, Chapter (3), which could be altered to increase the productivity of the machine. The next step used simulation results to determine how these variables influenced each other as well as the performance of the machine as a whole. The results of these simulations and their implications are discussed and listed in Chapter (4). In brief, they confirmed a need for a specialized processor subsystem designed for the task of comparing postings. Also, problem areas in the data storage and transfer systems were identified and solutions evaulated. These solutions included faster disk types and larger transfer bandwidths between secondary storage and main memory.

In Chapter (4) architecture (i) was postulated as a possible forerunner to more sophisticated retrieval machines. This architecture uses a secondary storage subsystem comprised of devices which are already commonly used by computer installations. With the aid of a specialized processor this machine yielded relatively good search times as well as an ability to run in small main memory sizes.

In Chapter (5) simulations and an analytical model were employed to study Architecture (i) in a multi-user environment. The performance variables measured included mean service time, user response time and search arrival rates. Different working conditions were tested by varying the job mix, the size of main memory and the number of storage devices on the data base channel. In short, we found Architecture (i) performed extremely well as compared to conventional standards.

In this paper we have studied possible machine architectures for a specialized information retrieval computer. The tools used to gather our performance data included simulations and an analytical model. With the knowledge obtained from these and other efforts related to the EUREKA Project, we will be able to better design efficient text retrieval systems.

LIST OF REFERENCES

- [1] L.A. Hollaar, et al, "System Architecture for Information Retrieval," submitted for publication.
- [2] J.R. Rinewalt, "Evaluation of some Features in a Full-Text Information Retrieval System," EUREKA Project Memo No. 141, Dept. of Computer Science, University of Illinois, Urbana, unpublished, 1975.
- [3] B.J. Hurley, "Inverted File vs. Full-Text Searching," EUREKA Project Memo No. 107, Dept. of Computer Science, University of Illinois, Urbana, unpublished, 1975.
- [4] W.H. Stellhorn, "A Specialized Computer for Information Retrieval," Ph.D. dissertation, Dept. of Computer Science, University of Illinois, Urbana, report UIUCDCS-R-74-637, 1974.
- [5] L.A. Hollaar, "A List Merging Processor for Inverted File Information Retrieval Systems," Ph.D. dissertation, Dept. of Computer Science, University of Illinois, Urbana, report UIUCDCS-R-75-762, 1975.
- [6] T.H. Martin, "A Feature Analysis of Interactive Retrieval Systems," Institute for Communications Research, Stanford University, Stanford California, report SU-COMM-ICR-74-1, 1974.
- [7] B.J. Hurley, "Scheduling a Head-Per-Track Disk and Special Processor for an Inverted File Retrieval System," EUREKA Project Memo No. 142, Dept. of Computer Science, University of Illinois, Urbana, unpublished, 1975.
- [8] National Library of Medicine, Master Mesh; November 1972.

- [9] J.M. Milner, "A Day in the Life of MED-LINE," EUREKA Project Memo No. 143, Dept. of Computer Science, University of Illinois, Urbana, unpublished, 1976.
- [10] J.A. White, J.W. Schmidt and G.K. Bennett, Analysis of Queueing Systems, Academic Press, New York, 1975

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-806	2.	3. Recipient's Accession No.
4. Title and Subtitle ANALYSIS OF COMPUTER ARCHITECTURES FOR INFORMATION RETRIEVAL		5. Report Date May 1976	
		6.	
7. Author(s) Bernard John Hurley		8. Performing Organization Rept. No. UIUCDCS-R-76-806	
9. Performing Organization Name and Address University of Illinois at Urbana-Champaign Department of Computer Science Urbana, Illinois 61801		10. Project/Task/Work Unit No.	
		11. Contract/Grant No. US NSF DCR73-07980 A02	
12. Sponsoring Organization Name and Address National Science Foundation Washington, D. C.		13. Type of Report & Period Covered Master's Thesis	
		14.	
15. Supplementary Notes			
16. Abstracts This paper is the result of an effort to support and supplement the development of a specialized information retrieval computer. EUREKA. presently being built at the University of Illinois, Urbana			
Our next step is to identify the design variables that exist within this basic architecture. These are the parameters that can be varied to configure specific retrieval machines. Finally, simulation results are studied to evaluate the effect the design variables have on each other as well as on the architecture as a whole. From these results we will develop an understanding of how to construct a			
17. Key Words and Document Analysis. 17a. Descriptors retrieval machine using information on the various speeds and capacities of its components.			
Computer architecture Disk systems File processing Information retrieval System configuration			
17b. Identifiers/Open-Ended Terms			
17c. COSATI Field/Group			
18. Availability Statement Release Unlimited		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 57
		20. Security Class (This Page) UNCLASSIFIED	22. Price

FEB 12 1981

UNIVERSITY OF ILLINOIS-URBANA



3 0112 047000812